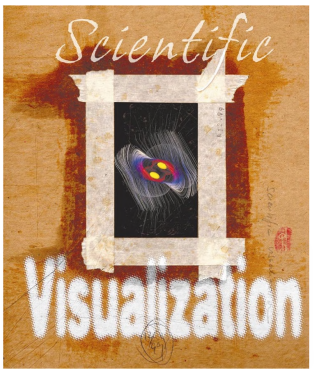
Cover Feature

Interactive Simulation and Visualization



As computational science and engineering applications grow in complexity, working with the data becomes increasingly difficult. An emerging technology—called *computational steering*—addresses this problem by providing a mechanism to integrate modeling, simulation, data analysis, and visualization.

Christopher Johnson
Steven G. Parker
Charles Hansen
Gordon L. Kindlmann
Yarden Livnat
Center for Scientific Computing and Imaging, University of Utah

Most of us perform data analysis and visualization only after everything else is finished, which often means that we don't discover errors invalidating the results of our simulation until postprocessing. A better approach would be to improve the integration of simulation and visualization into the entire process so that you can make adjustments along the way. We call this approach *computational steering*.

Computational steering is the capacity to control all aspects of the computational science *pipeline*—the succession of steps required to solve computational science and engineering problems. When you interactively explore a simulation in time and space, you *steer* it. In this sense, you can rely on steering to assist in debugging and to modify the computational aspects of your application.

Recently, several tools and environments for computational steering have begun to emerge. These tools range from those that modify an application's performance characteristics, either by automated means or by user interaction, to those that modify the underlying computational application. A refined problem-solving environment (PSE) should not only facilitate everything from algorithm development and performance tuning to application steering, but should also provide a rich environment for accomplishing computational science.

FACETS OF INTERACTIVE VISUALIZATION

One common method for visualization is to explore three-dimensional data sets by examining *isosurfaces*, three-dimensional surfaces representing the locations of a constant scalar value within a volume.

Visual isosurfaces

You can visualize isosurfaces in many ways, including geometric representation, direct isosurface rendering, and volume rendering. In the sidebar "Volume Visualization," we describe methods for achieving interactivity with the first method and also point out the power of volume rendering.

The standard, nonaccelerated method for isosurface extraction is the Marching Cubes algorithm,[1] which checks each cell of the data set to see if it contains an isosurface. But using this algorithm means that you're checking many cells repeatedly, even when they don't contain data that contributes to the final image.

To avoid unnecessary cell checking, you can preprocess the data and build specific data structures that let you rapidly extract isosurfaces. One such method is the Near Optimal IsoSurface Extraction (NOISE) algorithm.[2] Using a new representation of the underlying domain, called the *span space*, the isosurface extraction algorithm relies on a worst-case complexity parameter that reduces the search domain for a particular isosurface from all the cells to only those cells that contain an isosurface.[2,3] Figure 1 shows a screen shot from an interactive parallel rendering system built with the NOISE algorithm.[4]

While algorithms such as NOISE have effectively eliminated the search phase bottleneck, the cost of constructing and rendering an isosurface remains high. Many of today's simulation data sets contain very large, complex isosurfaces that can easily overwhelm even state-of-the-art graphics hardware.

Complex isosurfaces

These massive isosurfaces generally have two characteristics:

- many of the polygons that make up the isosurface are smaller than a pixel, and
- these polygons have a significant amount of depth complexity, which means that the number of polygons represented by a single pixel is quite high.

Authorized licensed use limited to: The University of Utah. Downloaded on August 28, 2009 at 18:20 from IEEE Xplore. Restrictions apply.
0018-9162/99/$10.00 © 1999 IEEE

Volume Visualization

In direct volume rendering, optical properties, like color and opacity, make the individual values in the data set visible. The appeal of direct volume rendering is that no intermediate geometric information needs to be calculated, so the process maps from the data set directly to an image.

In practice, you have to do a significant amount of work to create an intelligible rendering. The basic problem is to create the mapping from data values to optical properties. This process, called the *transfer function*, requires selecting those aspects of the data set to appear in the rendering. You generally find a good transfer function only after a slow process of trial and error.

Visualizing scanned medical data

Our work on this problem focuses on a specific use of direct volume rendering: visualizing scanned medical data to display the surfaces of organs or bone. While showing these boundaries may seem to be a problem of edge detection (a problem, in other words, of computer vision), the two problems differ in a subtle but important way. While edge detection seeks to locate edges within the two or three spatial dimensions of an image, we need to locate boundaries within the range of data values occurring in the data set, values which represent some physical property like radio-opacity or proton density. It is the lack of a spatial component to this process that makes it unintuitive.

Our solution is to create the transfer function in two distinct steps. The first step requires performing an automated analysis using metrics borrowed from edge detection; we then project the results into the space of data values. We use the information accumulated this way to compute a *distance function*, which gives the signed distance to an object boundary as a function of data value. Having the distance function simplifies the user's task immensely, because it can constrain the otherwise unwieldy parameter space of transfer functions to only those that emphasize the boundaries within the data set. In the second step, the user generates a *boundary appearance function*, which maps distance (rather than data value) to color and opacity.

Essentially, you create the transfer function as a composite of the distance function and the boundary appearance function. Doing so allows you to retain control over the transfer function but at a comfortable distance from the underlying space of raw data values.

More complex rendering

The two-step approach has immediate relevance to more complex rendering tasks. The domain of the transfer function may not be just the one-dimensional range of data values, as described here, but some higher dimensional feature space. As long as a distance function can be computed, the same boundary appearance function can used to create the transfer function. One example of such a domain is the two-dimensional space of data value and gradient magnitude.

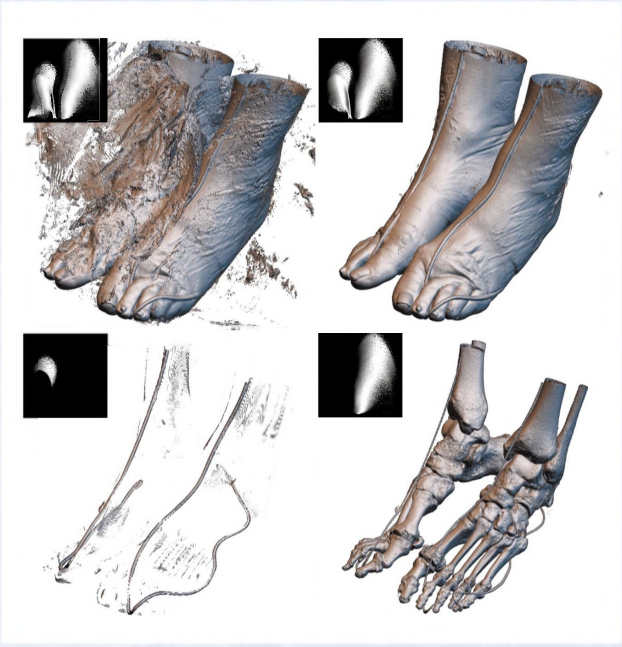


Figure A. Manipulation of an automatically generated two-dimensional opacity function to selectively render different material boundaries: skin (upper right), bone (lower right), and the registration cord laced around the body prior to scanning (lower left).

Figure A shows volume renderings from the visible woman data set made with two-dimensional opacity functions, shown here as the inset gray-scale images. The horizontal and vertical axes represent data value and gradient magnitude; brightness represents opacity. If the initially generated opacity function shows more features than desired (upper left), the structure of the opacity function makes it simple to select individual boundaries for rendering.

Abstracted levels of interaction become more important as the size of data sets—and hence required rendering time—grow with advances in measurement equipment and techniques. Also, where data sets and associated volume-rendering methods are more complex, methods for guiding the user toward useful parameter settings become a necessary part of generating informative visualizations. Research in these areas is currently under way.

By reducing the required generation of nonvisible isosurfaces, we can increase the interactivity for such data sets.

Recall that Marching Cubes examines every cell and that search acceleration methods reduce that search to only those cells containing an isosurface. If we can further reduce the search to only those isosurfaces that are visible in the final image, we can gain more interactivity. Since single pixels represent many polygons, we can achieve this goal by using a view-dependent algorithm.

The Wise algorithm,[5] for example, prunes sections of data that are visually obscured by sections of the isosurface already extracted. The algorithm works these tests in software against a one-bit-per-pixel virtual screen and then forwards the triangulation of the visible cells to the graphics accelerator for rendering by the hardware. It is at this stage that the PSE resolves the final graphics.

This kind of work explores the middle ground between a mostly hardware-based algorithm (like Marching Cubes) and a purely software-based algo-

Authorized licensed use limited to: The University of Utah. Downloaded on August 28, 2009 at 18:20 from IEEE Xplore. Restrictions apply.

rithm (like ray-tracing). The goal in doing so is to reduce the load on the network and graphics hardware by performing some of the visibility tests in software.

This approach can lead to an output-sensitive method that reduces the load of other components in the visualization pipeline,[6] such as transmission of the isosurface geometry over a network.

FACETS OF COMPUTATIONAL STEERING

One of the primary goals of computational steering is to make scientific applications more interactive and flexible. Unfortunately, many codes are neither intuitive nor flexible. Adapting complex, rigid applications to computational steering methods can be difficult. To complicate matters, many steering systems force you to adopt a particular methodology or rely on a particular software tool.

Such rigidity is especially undesirable given the long life span of most scientific applications and the wide range of computational requirements found in these applications. Before we can achieve acceptable interaction and flexibility, however, we need to address four facets of the problem:

- control structures,
- data distribution,
- data presentation, and
- user interfaces.

These facets may not all be present in every problem—and they don't portray the entire problem—but they do outline fundamental considerations.

Effective systems must foster efficient extraction of relevant scientific information, from simple *x-y* plots to sophisticated three-dimensional visualizations. This kind of efficiency requires that the system be tightly coupled with the simulation code to provide more information than would normally be available in a separate data analysis system.

The programming model

A computational steering *programming model* describes the software architecture that integrates computational components to extract information efficiently and permit changes to simulation parameters and data. This new architecture often requires some modification of the original scientific code, but the extent and nature of the changes will depend on the model you choose.

At one extreme, you will have to rewrite the scientific program to support steering. Less radical approaches may reuse pieces of the computation or use off-the-shelf visualization packages to simplify the construction of a steerable system.[7,8]

The means by which a system permits you to specify various types of changes to be made in the simulation is critical. For example, various devices may allow you to specify changes in the computation that range from simple text files to sophisticated scripting languages to graphical user interfaces. Having flexibility in each of these areas can mean the difference between efficient and inefficient computation.

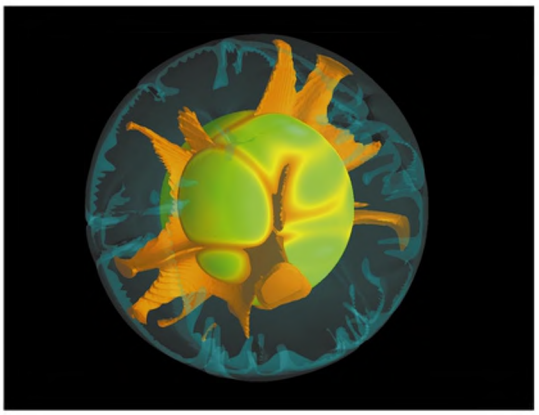


Figure 1. An earth mantle convection system showing multiple isosurfaces extracted with the Near Optimal IsoSurface Extraction (NOISE) algorithm.

Another less obvious issue is that of integrating changed data into the simulation in a scientifically meaningful fashion. In most coupled systems, it does not make sense to change one quantity without making corresponding changes in others. For example, in a fluid dynamics system, it would not make sense to allow sudden changes in pressure without making corresponding changes in another quantity, like temperature, to maintain balance for the ideal gas law.

Several approaches can be used to make scientific applications into steerable systems. Each approach has strengths and weaknesses. In many cases, you could use components of all of these approaches.

Program instrumentation

One way to implement steering is to make small modifications in the source to provide access points for the parameters and results. This process is called *instrumentation* and typically takes the form of subroutine calls inserted in the code wherever results become available or when new parameters can be used.

These calls can transmit data to and from a separate visualization process. They might perform visualization tasks internally or might trigger a thread that siphons the data while the computation continues concurrently. Systems such as Falcon[9] and Progress[10] rely on this approach.

The instrumentation technique has the advantage of being minimally intrusive to an existing scientific code. Instrumentation works well for domain-specific applications and development of new applications when you clearly define your parameters.

However, it may provide only limited control over the existing applications, as you may access only the

Authorized licensed use limited to: The University of Utah. Downloaded on August 28, 2009 at 18:20 from IEEE Xplore. Restrictions apply.

Contributing to Medical Science

Much of our research focuses on SCIRun,[1,2] an environment we use to create and steer scientific applications. SCIRun can construct an application by connecting computational elements to form a program, which can contain several computational elements as well as several visualization elements, all of which work together to orchestrate a solution. You can change inputs and parameters interactively in SCIRun and you can get immediate feedback on these changes.

Simulating the human thorax

Every year, approximately 500,000 people die suddenly because of abnormalities in their hearts' electrical systems and from coronary artery disease. While medical professionals have used external defibrillation units for some time, their use is limited because it takes such a short time for a heart attack victim to die from insufficient oxygen to the brain.

Lately, researchers have begun to look for a practical way of implanting electrodes within the body to defibrillate a person automatically upon onset of cardiac fibrillation. Because of the complex geometry of the human thorax and the lack of sophisticated thorax models available to researchers, most work on defibrillation devices has relied on animal studies. In order to provide an alternative to animal testing, we constructed a large-scale computer model of the human thorax, the Utah Torso Model.

Using the Utah Torso Model, we can simulate a multitude of electrode configurations, electrode sizes, and magnitudes of defibrillation shocks. Given the large number of possible external and internal electrode sites, magnitudes, and configurations, it is a daunting problem to test and verify various configurations.

Measuring brain voltage

Excitation currents in the brain produce an electrical field that can be detected as small voltages on the scalp. By using electroencephalograms (EEGs) to measure changes in the patterns of the scalp's electrical activity, physicians can detect some forms of neurological disorders. However, these measurements provide physicians with only a blurred projection of brain activity. A pervasive problem in neuroscience is determining which regions of the brain are active, given voltage measurements at the scalp.

If accurate solutions to such problems could be obtained, neurologists would gain noninvasive access to patient-specific cortical activity. Access to such data would ultimately increase the number of patients who could be effectively treated for neural pathologies such as multifocal epilepsy.

Putting SCIRun to work

We use SCIRun to solve these two bioelectric field problems in medicine. In the first case, illustrated in Figure B, we use SCIRun to design internal defibrillator devices and measure their effectiveness in an interactive graphical environment. In the second case, we use SCIRun to develop and test computational models of epilepsy and localize the focus of electrical activity within the brain.

Since trial and error has in the past determined placement of the electrodes for either type of defibrillator, one of our goals is to use SCIRun to assist in determining the optimum electrode placement to terminate fibrillation. Figure C shows an image of our algorithm running within the SCIRun environment. By interacting with the model, the user can seed the algorithm with sources placed in physiologically plausible regions of the model. In practice, this computational steering capability translates into more than a 50 percent reduction in the number of iterations required.

Saving resources

Even simulated experiments of such complexity can be time and cost prohibitive. By using SCIRun, however, you can interactively steer such a simulation, which can have a direct impact on both time and cost.

Long before the system completes a detailed solution, you might determine that the configuration isn't acceptable and might therefore try a new configuration and restart the simulation. Instead of throwing everything away and starting over, SCIRun

parameters that you've instrumented. This technique also has implementation complications, such as data transmission overhead if you send data to a separate visualization process, computation stalling if you do visualization internally, and complicated synchronization between visualization and computation components.

Directed scientific computation

An alternative approach to program instrumentation is to break a code up into various modules that you control explicitly by issuing a sequence of commands. One popular approach for doing this is to rely upon scripting languages such as Python or Tcl. Many developers have used this model successfully in commercial packages such as IDL, Matlab, or Mathematica. And large labs, like Los Alamos and Lawrence Livermore National Laboratories,[7,11] use this model for physics applications.

By using this method you can reuse almost all of the original scientific code. Also, this method is quite easy to implement on most systems, since it avoids the problems of managing multiple threads and synchronization. This makes it suitable for controlling most kinds of applications—including large parallel applications on both distributed-memory and shared-memory systems.

The scripting language interface also provides expert users with a fine degree of control over most, if not all, program parameters. This means that the system can be used for scripting long-running simulations, prototyping new features rapidly, and even debugging.

As an added benefit, most scripting languages provide access to Tk, a toolkit for building graphical user interfaces. Thus, you could implement a graphical user interface over a command-driven system if you need to.

Dedicated steering systems

If you're fortunate enough to be designing a new model with computational steering in mind, you'll have more room for innovation. For example, we designed the SCIRun system specifically for compu-

Authorized licensed use limited to: The University of Utah. Downloaded on August 28, 2009 at 18:20 from IEEE Xplore. Restrictions apply.

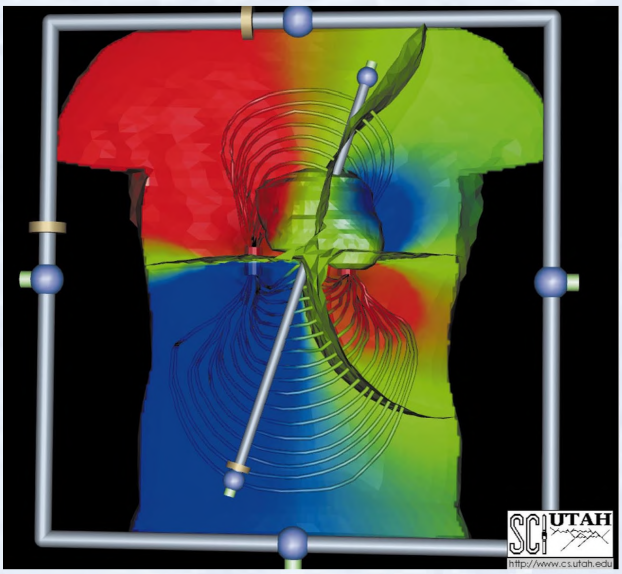


Figure B. A SCIRun three-dimensional rendering of an internal defibrillation simulation. A user can interactively investigate the voltage surfaces and the lines of electric current flow as well as change the position and shape of the electrode source.

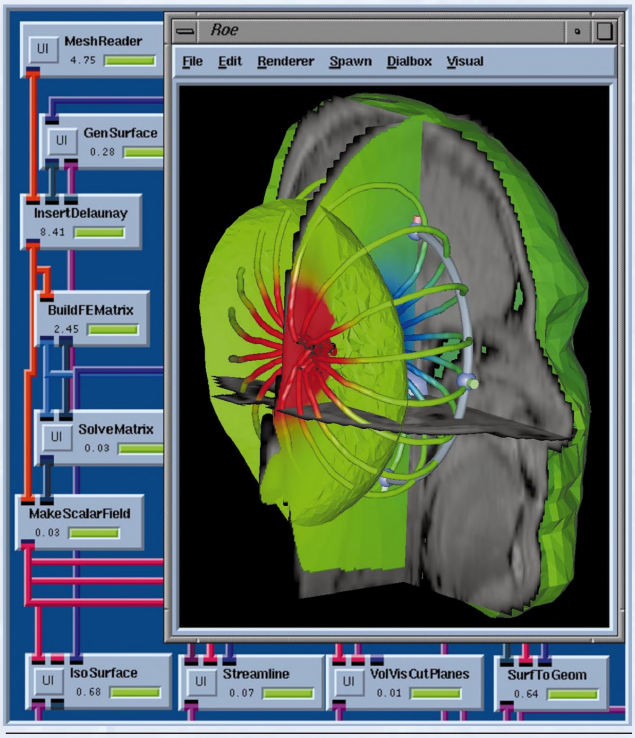


Figure C. A SCIRun EEG rendering. A user can select physiologically plausible regions of the model in which to seed the algorithm, thereby steering the algorithm to a more rapid convergence.

uses temporal and spatial coherence to compute only those aspects that have changed between the previous simulations and the new one. In scenarios where you make only small changes to input parameters, you can achieve a significant CPU and storage savings.

There are many engineering design problems that could benefit from such a system. These range from the biomedical problems discussed here to traditional mechanical design.

References

1. S.G. Parker, D.M. Weinstein, and C.R. Johnson, "The SCIRun Computational Steering Software System," *Modern Software Tools in Scientific Computing*, E. Arge, A.M. Bruaset, and H.P. Langtangen, eds., Birkhäuser, Boston, 1997, pp. 1-44.
2. C.R. Johnson and S.G. Parker, "Applications in Computational Medicine Using SCIRun: A Computational Steering Programming Environment," *Proc. Supercomputer 95*, Springer-Verlag, New York, pp. 2-19.

tational steering. (See the sidebar "Contributing to Medical Science" for a description of how we're using SCIRun to solve certain medical problems.) It allows you to construct a simulation using reusable computational components connected within a visual programming environment. We designed each of these components, in addition to the system as a whole, to integrate modeling, computation, and visualization, and to facilitate the interactive steering of all phases of the simulation.

Unlike directed approaches, the SCIRun system manages each module as an independent execution thread, which allows you to interact with the system even during long-running operations—although data dependencies between operations may force you to wait for results. The dataflow model used by SCIRun also simplifies synchronization issues, making it relatively easy to siphon data and feed it to various analysis and visualization modules as a simulation progresses.

A dedicated steering system such as SCIRun offers many advanced features over traditional systems, especially if you're writing a new application. While such systems offer a somewhat flexible model that can accommodate existing code, applying them to existing code can be difficult in practice. Scientific applications may not have been written in a manner that is easily translated to such a steering environment. In other cases, there may just be too much dust on the deck, in which case it would be easier just to start over.

Advanced steering systems like SCIRun may also rely on a style of programming that is difficult to implement on certain machines. For example, the computational model used in SCIRun was designed for use with shared-memory symmetric multiprocessing systems. Implementing the system on a distributed message-passing machine involves different challenges.

Data presentation

Another consideration is presenting the information itself. Choices range from off-the-shelf visualization tools to custom analysis and visualization programs.[7,8,12] Ideally, a flexible control structure would allow a variety of tools to be mixed and matched for each specific problem.

Authorized licensed use limited to: The University of Utah. Downloaded on August 28, 2009 at 18:20 from IEEE Xplore. Restrictions apply.

Even though it seems like computational steering is well poised for widespread popularity, there are still a few barriers.

Often, the full power of a computational steering system comes from the tight integration of scientific codes with visualization tools that were designed for that problem. You would typically use visualization to view the results of a computation, but it can be used in other roles as well, such as to visualize memory usage, algorithm performance, or multiprocessor communication patterns. And you can also use visualization to examine intermediate results, matrix structures, mesh details, or domain decompositions.

In other cases, it may be possible to use preexisting software as a presentation mechanism. For example, a system may use public domain graphing libraries and image processing tools. Researchers have also used commercial systems with success.[7]

Putting it all together

Because computational scientists have a wide variety of needs, computational steering systems should be able to serve different users and applications by operating in different modes. For example, in a debugging or development mode, you may want to use a highly interactive system that allows parameters and simulations to be run in almost real time. However, you may later want to run large simulations requiring hundreds of CPU hours. In this case, it may be easier to use a control language and write batch scripts than to use an interactive environment.

You may also want to write extensions or use the system with your own applications. Unfortunately, doing so may require a detailed understanding of each facet of the problem. Because researchers can see the complexity of applying such systems to their own work, many continue to ignore computational steering efforts altogether. While there is no easy answer, it is clear that this issue will need to be addressed in order for steering to be adopted into mainstream scientific computing efforts.

STEERING EVOLUTION

The convergence of high-end graphics workstations and supercomputers will allow more people to perform both computations and visualizations. Instead of transferring gigabytes of data to a visualization workstation, you'll be able to perform the visualization directly on a supercomputer.

Also, while most people did not have ready access to graphics workstations a few years ago, today they do. The popularity of advanced graphics cards means that high-powered workstations are becoming fairly commonplace.

While simulation is becoming more accessible, it is also playing an increasingly important role in today's scientific and engineering worlds. Increased safety responsibilities, heightened environmental awareness, and cheaper CPU cycles have all increased the motivation for many engineers to do simulation instead of real-world experiments.

Even though it seems like computational steering is well poised for widespread popularity, there are still a few barriers. For example, many supercomputers are currently set up for batch processing, not for interactive use. A computational steering system violates many of the assumptions on which a batch mode system relies. As a result, site managers and system developers need to recognize the benefits of steering large-scale applications before they'll implement the technology.

Even though spending some time interactively setting up a simulation may save tens to hundreds of hours of production time, the preconception that interactivity wastes CPU cycles will remain difficult to overcome. Performing visualization and rendering tasks using these cycles can be extremely expensive, particularly since accounting systems at supercomputer sites focus on CPU cycles used rather than on those saved.

Note, however, that costs are important to consider for large problems run on supercomputers, but may not be relevant to the scientist doing smaller problems on a desktop superworkstation. Furthermore, with the trend toward smaller machines placed at the supercomputer center site, you can offload tasks from the primary machines at large facilities, which means you can reduce cycle costs up front.

FUTURE DIRECTIONS

Interactive simulation and visualization will succeed only if such systems can be useful to scientists and engineers. These systems need to be

- modular and easy to extend with existing code,
- upgradeable without encountering complex programming issues,
- adaptable to hardware ranging from the largest of supercomputing systems to low-end workstations and PCs, and
- demonstrably usable in scientific research.

Working prototypes are a start, but we hope that scientists and engineers will soon view steering systems as among the most useful and necessary of their working tools.

We are currently researching distributed steering systems that would involve multiple platforms and languages. Ultimately, we hope that this research will lead to the development of steering systems that are

- highly interactive and easy to extend,
- capable of working well with large amounts of data, and

Authorized licensed use limited to: The University of Utah. Downloaded on August 28, 2009 at 18:20 from IEEE Xplore. Restrictions apply.

- adaptable to ordinary workstations, high-end servers, and supercomputing systems.

Furthermore, while component-oriented designs have achieved success in many areas, making such component-oriented designs work for interactive computational steering systems continues to be a challenge.

Other advances in simulation technology, such as the increased importance of adaptive mesh refinement, present both a challenge and an opportunity for computational steering. While an adaptive structure is more difficult to manage, it presents a natural vehicle to perform multiresolution data transmission and visualization, and even to effect user-directed changes in the computation.

We hope that as the community addresses and solves these problems, emphasis will shift toward issues related to data management, quality of service, and reproducibility of results. ❖

Acknowledgments

This research was supported in part by awards from the DOE, NSF, and NIH. We recognize the valuable resources available at the SGI Utah Visual Supercomputing Center at the University of Utah and we thank David Weinstein for his helpful comments and suggestions.

References

1. W.E. Lorensen and H.E. Cline, "Marching Cubes: A High Resolution 3D Surface Construction Algorithm," *Computer Graphics*, July 1987, pp. 163-169.
2. Y. Livnat, H. Shen, and C.R. Johnson, "A Near Optimal Isosurface Extraction Algorithm Using the Span Space," *IEEE Trans. Vis. Comp. Graphics*, IEEE CS Press, Los Alamitos, Calif., 1996, pp. 73-84.
3. H.W. Shen et al., "Isosurfacing in Span Space with Utmost Efficiency," *Proc. IEEE Visualization 1996*, IEEE CS Press, Los Alamitos, Calif., pp. 287-294, 1996.
4. J. Painter, H.P. Bunge, and Y. Livnat, "Case Study: Mantle Convection Visualization on the Cray T3D," *Proc. IEEE Visualization 1996*, IEEE CS Press, Los Alamitos, Calif., 1996, pp. 409-412.
5. Y. Livnat and C.D. Hansen, "View Dependent Isosurface Extraction," *Proc. IEEE Visualization 1998*, IEEE CS Press, Los Alamitos, Calif., 1998, pp. 175-180.
6. W. Schroeder, K. Martin, and W. Lorensen, *The Visualization Toolkit: An Object-Oriented Approach to 3D Graphics*, Prentice Hall, New York, 1997.
7. P.M. Papadopoulos and J.A. Kohl, "A Library for Visualization and Steering of Distributed Simulations Using PVM and AVS," *High Performance Computing Symp.*, Montreal, Canada, 1995.
8. Y. Jean et al., "An Integrated Approach for Steering, Visualization, and Analysis of Atmospheric Simulations," *Proc. IEEE Visualization 1995*, IEEE CS Press, Los Alamitos, Calif., 1995, pp. 411-418.
9. W. Gu et al., "Online Monitoring and Steering of Large-Scale Parallel Programs," *Proc. 5th Symp. Frontiers of Massively Parallel Computing*, IEEE CS Press, Los Alamitos, Calif., 1995, pp. 422-429.
10. J. Vetter and K. Schwan, "Progress: A Toolkit for Interactive Program Steering," *Proc. 24th Int'l Conf. Parallel Processing*, IEEE CS Press, Los Alamitos, Calif., 1995, pp. 139-142.
11. P. Dubois, "Making Applications Programmable," *Computers in Physics*, Jan. 1994, p. 70.
12. D.M. Beazley and P.S. Lomdahl, "Lightweight Computational Steering of Very Large-Scale Molecular Dynamics Simulations," *Supercomputing 96*, ACM Press, New York, 1996.

Christopher Johnson *directs the Center for Scientific Computing and Imaging at the University of Utah. His research interests include inverse and imaging problems, problem-solving environments, computational steering, and scientific visualization. He received a PhD from the University of Utah.*

Steven G. Parker *is a member of the Center for Scientific Computing and Imaging research staff. His research interests include scientific visualization, computer graphics, user interaction, and scientific computing. He received a PhD in computer science from the University of Utah.*

Charles Hansen *is an associate professor of computer science at the University of Utah. His research interests include large-scale scientific visualization, massively parallel processing, parallel computer graphics algorithms, 3D shape representation, and computer vision. He received a PhD in computer science from the University of Utah.*

Gordon L. Kindlmann *is a graduate student in the department of computer science at the University of Utah. His research interests include volume rendering. He received an MS in computer graphics from Cornell University.*

Yarden Livnat *is a research associate in the department of computer science at the University of Utah. His research interests include computational geometry, scientific computation and visualization, and computer-generated holograms. He received a PhD in computer science from the University of Utah.*

Contact the authors through Chris Johnson at crj@cs.utah.edu.

Authorized licensed use limited to: The University of Utah. Downloaded on August 28, 2009 at 18:20 from IEEE Xplore. Restrictions apply.